

Controlling A Robot With a Personal Computer

An Honors Thesis (HONRS 499)

by

Jeffrey R. Carver

Thesis Advisor
Dr. Wesley Baldwin

A handwritten signature in black ink that reads "Wesley Baldwin". The signature is written in a cursive style with a large, stylized 'W' and 'B'.

Ball State University

Muncie, Indiana

May 2002

Abstract

This project involved creating an interface by which a pneumatic robot was controlled by a personal computer. The purpose of this project was to provide a more modern and convenient method for controlling a robot that previously had only manual electric switches for controls. The robot will be utilized for instructional purposes. Instructions for operating the robot with the new interface are included as well as a thorough explanation and schematic of how the interface works. Finally, the actual programming code used in both the micro-controller and the Visual Basic application implemented in the project is included in the appendices.

Acknowledgements

I would like to express my sincere appreciation to my thesis advisor, Dr. Wesley Baldwin, for providing the idea for this project. I also owe Dr. Baldwin gratitude for allowing me access to all the necessary equipment for my project and for offering advice and suggestions about my ideas. I would also like to thank the MET laboratory technician Tom Jarvis of the Purdue University School of Technology at Ball State, who provided a considerable amount of useful information about the equipment I used and who also prepared and maintained the robot for my use.

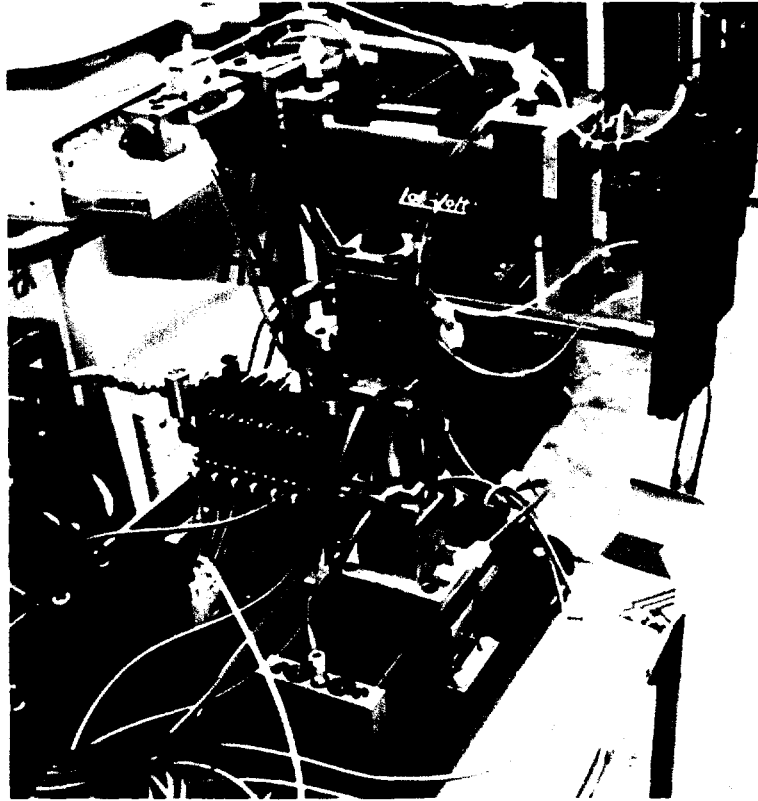


Figure 1

Introduction and Background

The Lab-Volt Model 6060 Pneumatic Robot (Figure 1) was manufactured in 1984 specifically for teaching students the basics of robotics and the electrical, pneumatic, and mechanical concepts involved in this field. This robot features six degrees of motion, which are controlled by six corresponding solenoid-controlled pneumatic valves. It is a point-to-point robot, meaning that when a degree of motion is triggered, air pressure moves the robot in that direction until a physical barrier is encountered; there are no velocity control, no stops in between positions, and the original controls for the robot consisted of a box containing six electric rocker switches (Figure 2) which are manually toggled to energize or de-energize the solenoid valves. This box is connected to the robot by means of a DB-9 connector identical to that of a nine-pin serial communications port.

The pneumatic cylinders that move the robot contain proximity switches, electronic sensors that indicate that the cylinder is either fully extended or fully retracted. In this way, the operator can determine through electronic means whether the requested motion has been completed after toggling the switch.

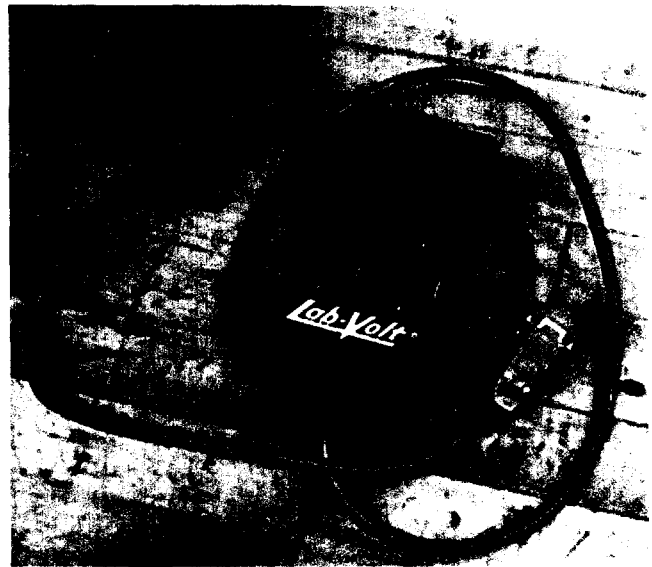


Figure 2

Objective

The objective of this project was to create a new interface between the Lab-Volt pneumatic robot and a personal computer so that an operator can control the robot's movements with simple mouse-clicks. Since attempts to obtain an updated interface from the manufacturer, who no longer supports this model of robot, proved futile, I was given the opportunity to create my own unique interface. While the new interface was primarily to provide means to control the robot through the personal computer, it would also be useful to monitor the states of the proximity switches at the ends of the pneumatic cylinders and communicate this information to the personal computer.

Component Selection

The first idea for controlling the robot through the personal computer was to implement a Programmable Logic Controller (PLC) which would communicate with the computer via a Direct Data Exchange (DDE) server and use its inputs and outputs to monitor and control the

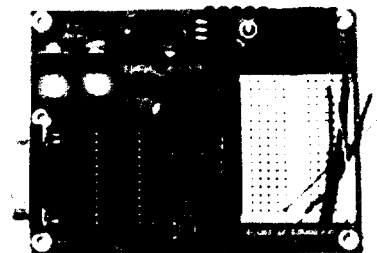


Figure 3

robot's motions. A DDE server can communicate directly with such applications as Microsoft Excel and Visual Basic programs. After obtaining pricing information on PLCs and DDE server software, it became clear a more economical option was necessary. Finally, I decided to implement a Basic Stamp 2 micro-controller (Figure 3). While the Basic Stamp 2 features input and output points like a PLC, it is also possible to attach a serial cable for direct communication with a personal computer, eliminating the need for costly DDE server software. On the other end of this communication, the personal computer, it is necessary to have software designed for serial communication. I decided to write a simple program of my own using Visual Basic because Visual Basic contains methods for programming serial communications and compiling programs into executable (.EXE) files.

The next area of component selection that needed to be addressed was finding a method to switch between the high-voltage signals used by the robot to the low-voltage signals used by the Basic Stamp 2, 120 volts AC and 5 volts DC, respectively. Clearly, the micro-controller signals are not powerful enough to actuate the robot, and the robot signals, applied directly to the micro-controller, would completely destroy it. To overcome this obstacle for controller-to-robot communications, I chose solid-state relays. These relays feature four terminals; two are for the positive and negative connections for a DC control voltage, and the other two are for bridging the AC load voltage. Only when the 5-volt control voltage is applied to one side does the switch on the other side close, completing the AC circuit and allowing current to flow. This way, the personal computer can communicate through the serial port to the controller, which then energizes the appropriate output with 5 volts DC, which is used to trip the relay and close the AC

circuit containing the corresponding solenoid. A feature of the solid-state relays I chose is optical isolation, meaning that the control voltage turns on a light that is detected by a sensor on the load side. In this configuration, the low-voltage and high-voltage circuits are entirely separated, preventing high voltage from arcing over and damaging the controller.

As for the robot-to-controller communication, the 120-volts AC needed to be converted into a 5-volt DC signal. The most obvious method for this was to implement a transformer and rectifier. These combinations are readily available in various voltages through electronics distributors as AC adapters. The only problem with the use of AC adapters is that, rather than simply wires, the inputs are the familiar blades for a wall outlet. I then obtained household extension cords as the simplest method for turning the blades into plain wires.

Establishing Serial Communication

The next step in the development of the interface was to establish communication between the controller and the personal computer through the serial cable. Since the actual serial port on the controller is primarily for uploading programs to the controller, the serial cable was instead connected to the input/output (IO) pins. In order to accomplish this, some modification was necessary to compensate for the fact that serial communication uses a 12-volt high and negative 12-volt low while the controller uses a 5-volt high and zero-

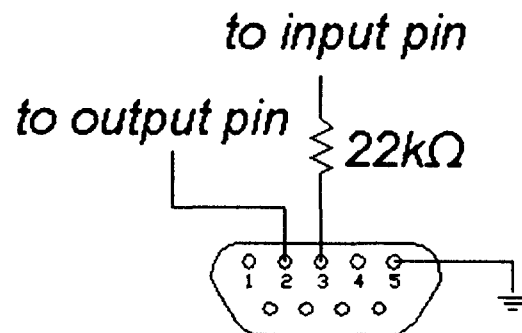


Figure 4

volt low for its IO pins. A $22k\Omega$ resistor must be placed in between the output pin of the serial cable and the IO pin of the controller, as shown in Figure 4 (BASIC 275). Having wired the cable to the controller, it was then necessary to program the controller to collect and interpret serial data. Obviously, in order to determine whether or not the controller is receiving serial data properly, something is needed that will send it serial data.

Therefore, at the same time I was attempting to write a functioning program for the controller, I was also trying to write one in Visual Basic. Running two separate but interacting programs at the same time added an entirely new dimension to debugging. Finally, by clicking the buttons on the Visual Basic program (Figure 5) with the mouse and using a multimeter to measure voltage across the outputs of the controller, I was able to determine that the outputs were indeed providing five volts when I pushed the button and zero volts after I pushed it again. Next, in order to test whether or not the controller-to-PC communication and the controller inputs were working properly, I routed one of the wires from the controller output into a controller input. In this arrangement, whenever I trigger the output, the voltage will go directly into the input, which should react accordingly. The Visual Basic program indeed received the data as was intended.

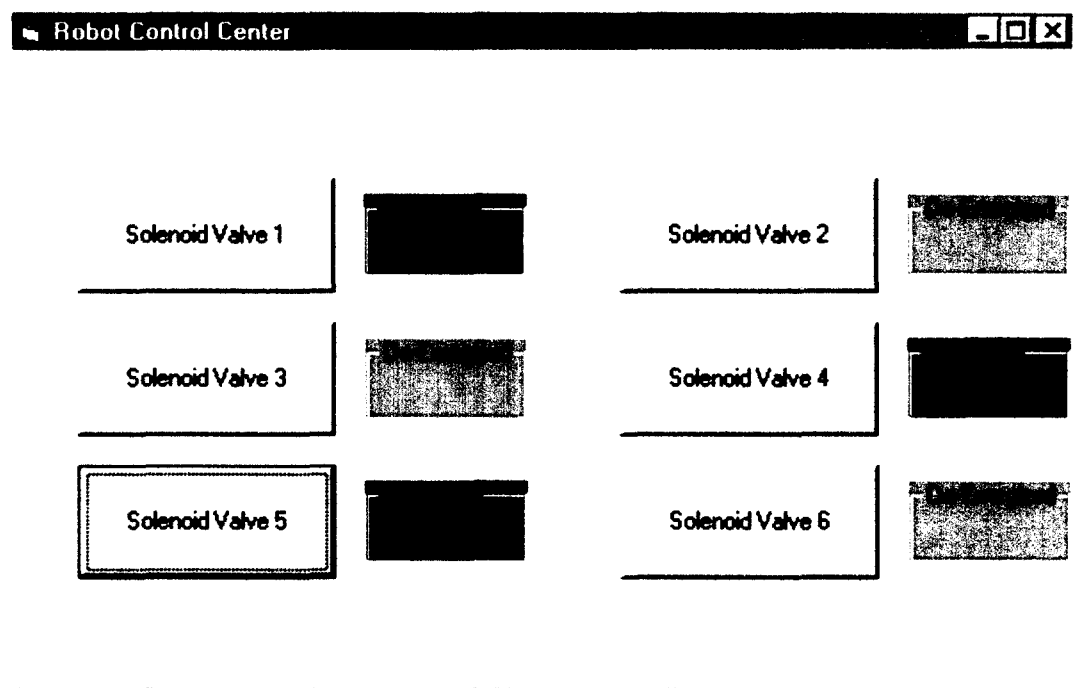


Figure 5

Explanation of the Programs

The text of the programming code for both the controller and the Visual basic program appear in appendices A and B, respectively. The codes I used to transmit the necessary information back and forth between the computer and the controller are simply two-digit numbers. The first digit represents the number of the input or output being used, and the second is either a 1 or 0, representing energized or not energized. A simple explanation of what exactly the controller code does is as follows:

- 1 Define the necessary variables, and place a label called "a"
- 2 Fill the "inputs" array with the appropriate numbers based on the states of the controller inputs.
- 3 Transmit through IO pin 14 the serial data contained in the "inputs" array, as decimal numbers.

- 4 Wait 1500 milliseconds for serial data to appear; if data does appear, continue forward; if it does not, go to label "a" (step 3 here).
- 5 Go to the appropriate label depending on the value of the serial data received.
- 6 Turn the appropriate output either high or low, and return to label "a" (step 3 here).

Unlike many higher-level programming languages, this version of Basic only supports labels as arguments for if-then-else statements. This is why the long list of labels were used, as opposed to simply directly executing the necessary actions from within the if-then-else statement. Another unusual drawback to the controller is that there is no serial buffer as personal computers have; that is, the controller can only collect serial data if it is currently being told to look for it, and while it is looking for serial data, it cannot do anything else. This is what necessitates the 1.5-second timeout while waiting for incoming serial data; if there were no timeout, the controller would not be able to send the serial data to the personal computer until after the next time the operator sent serial data to it by clicking one of the buttons on the Visual Basic program.

The Visual Basic program uses the same digit codes to determine the high or low status of the corresponding IO pin. Basically, this program sets up the serial communications using MSCOMM, waits until either the operator pushes a button or serial data arrives at the buffer, and then either sends the appropriate data to the controller through the serial port or changes one of the sensor indicators. Every time a button is pushed, the indicator to its right changes color between red and green, and its label changes between "energized" and "de-energized."

Of note in this section is the condition that to transport the robot and program to another personal computer, more than one file needs to be copied to the new personal computer. The "MSCOMM32.OCX" file needs to be copied from the "C:\Windows\System" folder to the same folder on the new machine so that the serial communications within the Visual Basic program will work properly. Microsoft provides this file for distribution with all Visual Basic programs that need it.

Problems with Shielding

Unfortunately, the feedback to the controller concerning the states of the sensors on the cylinders could not be implemented. This is because the unshielded 120-volt AC lines that deliver the signals to the robot generate too much interference and trigger the sensitive controller inputs at random, making monitoring them impossible. Therefore, the components of the programs pertaining to the sensors were omitted from the final versions of the programs.

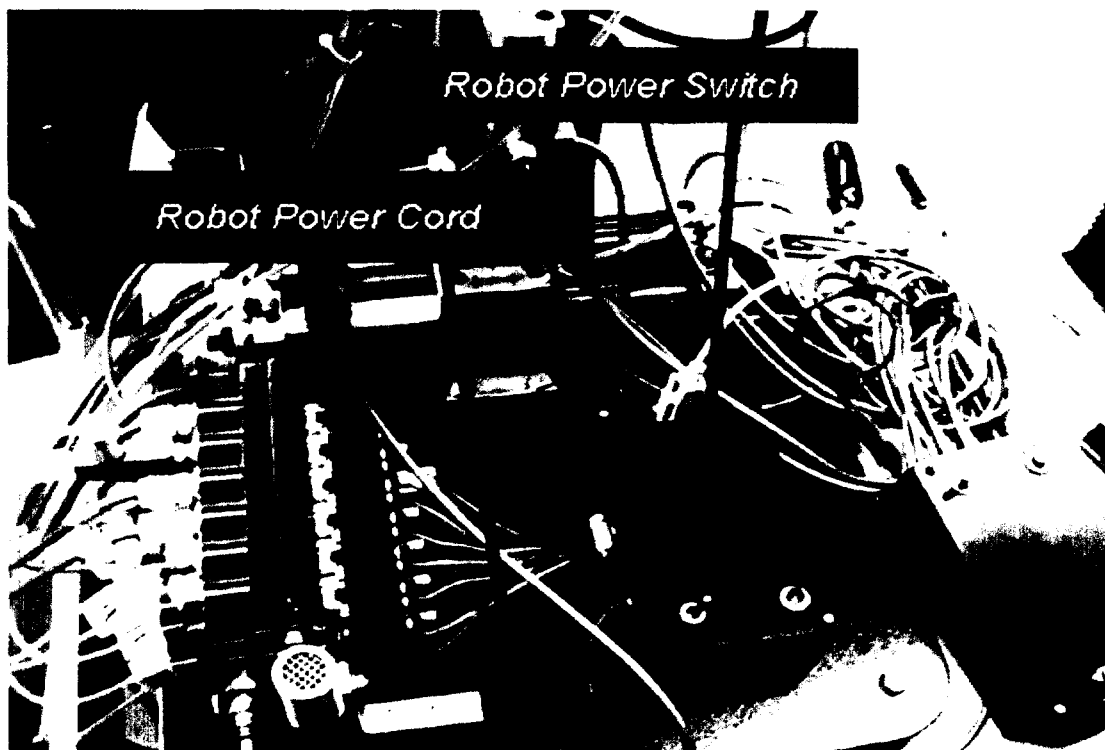


Figure 6

Instructions for Use

In order to use the robot with the new interface, first plug in the micro-controller if it is not already plugged in. It uses a standard AC adapter, which is plugged into the controller but not into an outlet until it is ready for use. Next, plug in the robot itself, which receives power through the black, three-pronged cord that also plugs into a standard 120-volt outlet; this is labeled “Robot Power Cord” in Figure 6. Now turn the robot on. To do this, flip the switch located on the box into which the black power cord returns, labeled “Robot Power Switch” in Figure 6. The red light next to the switch should now turn on. Then attach the airline to the robot as shown in Figure 7.

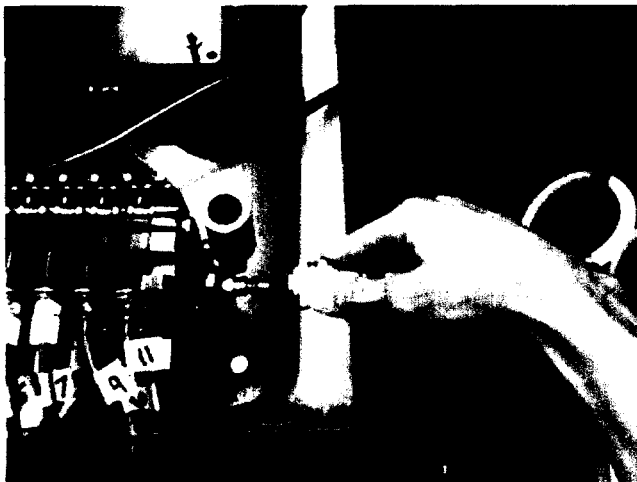


Figure 7



Figure 8

Now start the program, which is located on the desktop of the personal computer with the name “Robot Controls” and the icon shown in Figure 8. Double-click the icon, and the control window like the one shown in Figure 5 above will appear. Simply click the buttons to energize the corresponding solenoid valves and move the robot.

The manner in which the controller is connected to the robot can also be taken advantage of in other ways. The robot can be used for a specific purpose, on an

assembly line to move a part from one place to another, for example. Since the robot's solenoids are already wired to the controller outputs, the simple line of code "HIGH 1" will energize solenoid 1 and cause the robot's first degree of motion to actuate. Likewise, "LOW 1" will de-energize the solenoid and allow that particular motion of the robot to return to its original position. Another useful command for this application is the "PAUSE" command. Followed by a number, this command causes the controller to do nothing for that number of milliseconds. This allows wait time for the cylinders of the robot to actuate completely before the next motion is carried out. An example program could be as follows:

```
start:
HIGH 1
PAUSE 1200
HIGH 4
PAUSE 2300
HIGH 6
PAUSE 400
LOW 1
PAUSE 1000
LOW 4
PAUSE 1800
LOW 6
PAUSE 5000
GOTO start
```

When this example program is uploaded to the controller, solenoid 1 will be energized; the controller will wait for 1.2 seconds and then energize solenoid 4, and after another 2.3 seconds, it energizes solenoid 6. Then, after waiting 0.4 seconds, solenoid 1 will be de-energized; the controller will wait one second and de-energize solenoid 4, and finally, it de-energizes solenoid 6 after a 1.8-second wait. Having completed this, the controller then waits 5 seconds and repeats the process. Using methods such as this, any sequence and timing of operations of the robot can be achieved with relatively simple

programming, no matter how many steps. Simply plug a serial cable into the serial port on the back of the personal computer and the other end to the serial port on the Basic Stamp. Open the STAMPW.EXE program, type the code, and hit the “run” button. Once finished with this particular program, use the STAMPW.EXE program to open the controls.bs2 file, and then press “run” once more to return to the personal computer interface operation.

Appendix A: The Basic Stamp Program (controls.bs2)

[Italicized lines were removed in final version since controller inputs do not work]

```
serial VAR Byte
 VAR Byte(6)
i VAR nib
```

```
a:
```

```
inputs(0) = 10 + in8
(1) = 20 + in9
(2) = 30 + in10
(3) = 40 + in11
(4) = 50 + in12
(5) = 60 + in13
```

```
FOR i = 0 to 5
SEROUT 14, 16780, [DEC inputs(i)]
next
```

```
SERIN 15, 16780, a, 1500, a, [DEC serial]
```

```
IF serial = 10 then ser10
IF serial = 11 then ser11
IF serial = 20 then ser20
IF serial = 21 then ser21
IF serial = 30 then ser30
IF serial = 31 then ser31
IF serial = 40 then ser40
IF serial = 41 then ser41
IF serial = 50 then ser50
IF serial = 51 then ser51
IF serial = 60 then ser60
IF serial = 61 then ser61
```

```
goto a
```

```
ser10:
low 1
goto a
```

```
ser11:
high 1
goto a
```

Appendix A: The Basic Stamp Program (controls.bs2) [continued]

ser20:
low 2
goto a

ser21:
high 2
goto a

ser30:
low 3
goto a

ser31:
high 3
goto a

ser40:
low 4
goto a

ser41:
high 4
goto a

ser50:
low 5
goto a

ser51:
high 5
goto a

ser60:
low 6
goto a

ser61:
high 6
goto a
goto a

Appendix B: The Visual Basic Program

```
Private Sub Form_Load()
```

```
MSComm1.RThreshold = 2
```

```
MSComm1.InputLen = 2
```

```
MSComm1.CommPort = 1
```

```
MSComm1.Settings = "2400,N,8,1"
```

```
MSComm1.PortOpen = True
```

```
End Sub
```

```
Private Sub M1_Click(Index As Integer)
```

```
If Frame1.Caption = "Energized" Then
```

```
    MSComm1.Output = "10 "
```

```
    Frame1.Caption = "De-Energized"
```

```
    Frame1.BackColor = 111111
```

```
Else
```

```
    MSComm1.Output = "11 "
```

```
    Frame1.Caption = "Energized"
```

```
    Frame1.BackColor = 222
```

```
End If
```

```
End Sub
```

```
Private Sub M2_Click(Index As Integer)
```

```
If Frame2.Caption = "Energized" Then
```

```
    MSComm1.Output = "20 "
```

```
    Frame2.Caption = "De-Energized"
```

```
    Frame2.BackColor = 111111
```

```
Else
```

```
    MSComm1.Output = "21 "
```

```
    Frame2.Caption = "Energized"
```

```
    Frame2.BackColor = 222
```

```
End If
```

```
End Sub
```

```
Private Sub M3_Click(Index As Integer)
```

```
If Frame3.Caption = "Energized" Then
```

```
    MSComm1.Output = "30 "
```

```
    Frame3.Caption = "De-Energized"
```

```
    Frame3.BackColor = 111111
```

```
Else
```

```
    MSComm1.Output = "31 "
```


Appendix B: The Visual Basic Program [continued]

```
    Frame3.Caption = "Energized"
    Frame3.BackColor = 222
End If
End Sub
Private Sub M4_Click(Index As Integer)
If Frame4.Caption = "Energized" Then
    MSComm1.Output = "40 "
    Frame4.Caption = "De-Energized"
    Frame4.BackColor = 111111
Else
    MSComm1.Output = "41 "
    Frame4.Caption = "Energized"
    Frame4.BackColor = 222
End If
End Sub
Private Sub M5_Click(Index As Integer)
If Frame5.Caption = "Energized" Then
    MSComm1.Output = "50 "
    Frame5.Caption = "De-Energized"
    Frame5.BackColor = 111111
Else
    MSComm1.Output = "51 "
    Frame5.Caption = "Energized"
    Frame5.BackColor = 222
End If
End Sub
Private Sub M6_Click(Index As Integer)
If Frame6.Caption = "Energized" Then
    MSComm1.Output = "60 "
    Frame6.Caption = "De-Energized"
    Frame6.BackColor = 111111
Else
    MSComm1.Output = "61 "
    Frame6.Caption = "Energized"
    Frame6.BackColor = 222
End If
End Sub

Private Sub Form_Unload(Cancel As Integer)
MSComm1.PortOpen = False
End Sub
```

Bibliography

BASIC Stamp Programming Manual. Version 2.0c. Parallax, Inc. 2000.

Hoylman, Jared. "Understanding and Using Visual basic Part 5." Reynolds Electronics.

13 March, 2002. <http://www.rentron.com/sending_data.htm>

Schwaller, Anthony E. Model 6060 Pneumatic Robot Instructor Guide. St. Cloud State University. St. Cloud, Minnesota 1984.